Final Report:
NASA Grant NAG 3-1472
Parallelization of the Implicit RPLUS Algorithm

by

Dr. Paul D. Orkwis
University of Cincinnati

Abstract

The multiblock reacting Navier-Stokes flow solver RPLUS2D was modified for parallel implementation. Results for non-reacting flow calculations of this code indicate parallelization efficiencies greater than 84% are possible for a typical test problem. Results tend to improve as the size of the problem increases. The convergence rate of the scheme is degraded slightly when additional artificial block boundaries are included for the purpose of parallelization. However, this degradation virtually disappears if the solution is converged near to machine zero. Recommendations are made for further code improvements to increase efficiency, correct bugs in the original version, and study decomposition effectiveness.

Introduction

Computational Fluid Dynamics (CFD) is a field that continues to expand with each increase in computer technology. The development of faster, larger memory computers has allowed engineers to calculate solutions of bigger and more complicated flow fields. Parallel computers are the latest extension of this technology. Machines of this type use multiple processors to perform more than one computation concurrently. Computers exist which process multiple data streams using the same instructions or multiple data streams with different instructions. The total operation counts of these machines are scaled not only by processor speed but also by the number of available processors. However, the best approach to employing parallel machines with CFD solvers is not always readily apparent.

Explicit CFD codes are very amenable to parallel machines, but suffer in such applications as viscous chemically reacting flows because of the inherently restrictive Courant-Friedrichs-Lewy (CFL) stability condition. High grid resolution and stiff chemical reaction source terms require extremely small time steps for stable calculations. Implicit CFD solvers, such as the NASA Lewis RPLUS code [1,2], are popular because they reduce or eliminate the CFL restriction, allowing the user to employ a time step more closely related to the physical mechanisms of interest. However, explicit schemes allow a variety of approaches to parallelization at both the fine and coarse grain levels. One can efficiently parallelize at the loop level (requiring significant person-hours) or at a macro or block level (which is convenient for codes already employing a multi-block strategy). On the other hand, implicit schemes are not readily parallelizable in their original forms and must typically be modified at the

macro level using procedures such as *domain decomposition*.

In the domain decomposition approach, the solution domain is divided into subdomains that may be computed on different processors. Examples of this partitioning are easily found in the recent literature [3,4,5,6,7,8,9,10]. This approach is straightforward when applied to explicit schemes and the convergence rates of the methods do not suffer typically. The only complication with these schemes is message passing for the inclusion of out-of-block data. Conversely, implicit schemes change when decomposed, because the implicit matrix system solution must be split or altered when the domain is distributed among the processors. This sometimes leads to degradations in convergence rates, but is dependent upon the particular splitting employed and the problem being solved.

Domain partitioning at a macro level is not unique to parallel applications and has been used for some time in multi-block CFD algorithms for flow domains that do not have rectangular computational topologies. Simple applications of this strategy are backstep and cavity type geometries. Multi-block schemes can be parallelized naturally using domain decomposition techniques because their code structure already incorporates block to block communication. An example of this parallelization is the current research which has led to a parallel version of the two-dimensional NASA Lewis RPLUS reacting Navier-Stokes flow solver, a code currently in use by NASA researchers and industry engineers in its serial form. Multi-block codes of this type allow considerable decomposition flexibility but can be constrained by logical complexities if block interface conditions are communicated to more than one contiguous block per face. Flexibility must therefore be balanced against the inherent inefficiencies of accrued logic overhead. Contiguous block decompositions simplify this problem by requiring block faces to adjoin at most one other block face. One dimensional decompositions can also be used to simplify the logical overhead but suffer because they often reduce the flexibility of the potential decompositions, resulting in possible load balancing problems.

Parallelization, while straightforward for implicit multi-block algorithms, is not seamless. That is, the parallel and serial versions of a code may not behave in exactly the same manner. This is because the splitting generally requires more subdomains than one would naturally use for a complicated flow field, since the number of subdomains are determined by the available processors. The presence of the additional subdomain boundaries changes the matrix structure and can lead to reductions in convergence rates as information must be lagged at the block interfaces. This discrepancy is usually outweighed greatly by the advantages inherent in parallelization, but must be accounted for when considering parallelization options. In particular, systems with relatively small numbers of processors (like workstation clusters) are well suited to this approach.

Other difficulties with parallelization are the architecture and operating system variations between potential computing platforms. The considerable time investment for porting a serial code to a parallel machine or developing a parallel algorithm from scratch must be repeated in many cases because of these differences. Fortunately, this issue has been addressed in part by the appearance of parallelization protocols like PVM, MPI and APPL. These approaches define a standard set of commands that can be used when writing an application that will, in effect, translate the code to the actual commands of the current system. This allows the user to write one set of code and port it

almost indiscriminately to other parallel machines with available drivers. The use of the system then requires that a translation package be written once for that particular machine. Users then write commands in the protocol script and link to the translation software at compilation time. Of course, these protocols do require some overhead and will reduce somewhat the speed-up achieved by parallelization.

Our parallel version of the RPLUS code uses the MPI parallelization protocol. This protocol was chosen for many reasons such as efficiency, functionality and portability to many of the available computer architectures. Another very important reason for choosing MPI is that it is a standard. This means that all future releases of this protocol should be compatible with the old ones, hence a code written using an older version of MPI will not become obsolete if a newer version is released.

The MPI standard does not specify every aspect of a parallel program. Some aspects of parallel programming are left to the specific implementations such as process startup, scope of error handlers and the amount of system buffering provided for messages. Some of these specific implementations are: MPICH (developed by Argonne National Laboratory and Mississippi State University), LAM (developed by Ohio State University) and CHIMP (developed by Edinburgh Parallel Computing Center). Our parallel version of the RPLUS code uses MPICH version 1.0.12.

In the current work a parallel version of the implicit RPLUS2D solver was developed for application with the MPI protocol. An automatic two-dimensional domain decomposition approach was employed as a means of static load balancing. Residuals and convergence rate issues were briefly explored. The following sections describe the approach employed, give the details for running the algorithm, review the experimental approach used, discuss the findings of the research, and recommend future directions to be taken for improving the code.

Approach

The RPLUS2D code is an implicit reacting Navier-Stokes flow solver incorporating a multiple block strategy. This approach makes the code very amenable to a macro-level domain decomposition strategy as opposed to a finer grain parallelization at the loop level. The code is therefore geared toward parallel machines with relatively few processors, such as the Cray C-90 or workstation clusters. The current research capitalized on the block-block boundary data structures that are already in place in RPLUS for parallelization of the code. The data exchange at the block boundaries is implemented using the MPI_SENDRECV subroutine. This is a blocking subroutine, therefore, tight synchronization is achieved among the blocks. A flow chart of the parallel version of the RPLUS code is given in Figure 1.

Two options are included for domain decomposition, an automatic domain decomposition (ADD) approach and a user defined approach. Both approaches assume contiguous block faces. The ADD approach assumes only a Cartesian topology so far. Other topologies will hopefully be implemented in the future. In this approach, if the size of the domain is evenly divisible by the number of processors, then the domain is split into blocks of equal sizes. However, if the domain size is not

evenly divisible by the number of processors, then the domain is split in such a way to give optimal load balancing. This is done using the subroutine MPE_DECOMP1D which is included in this report. The user defined approach allows the code to be implemented with some external decomposition strategy. If this approach is chosen, then the user employs the original procedure in the RPLUS code to supply the bounds of each block explicitly.

It is important to recognize that since the former represents a static load balancing approach no mechanism exists to include load balancing alterations caused by inclusion of the reacting flow equations. This can be a problem because the ADD assumes equal work per node. This may not be the case if the reactions do not occur at all nodes or if work involved with the iterative determination of the reaction rate source terms is nonuniform. The current research does not address these issues.

The following changes were made to the code to enable parallelization and to enhance efficiency.

- The makefile for the original RPLUS code was replaced by a makefile for the parallel version. Compilation of the different pieces of the code and linking to the appropriate libraries is done in this file. Changes made to a certain part of the code will not necessarily require a complete recompilation.

- The file grid.dat is the same as that used in the original RPLUS code; no changes have been made.

- The file mpif.h is necessary in every MPI Fortran program and subprogram to define various constants and variables. This file comes with the MPI implementation MPICH.

- The file PAR.F contains only a parameter statement to define the size of the problem, number of species, number of reaction steps and number of blocks. The user should be sure to change this file when changing any of these parameters. A complete recompilation will be required when this file is changed.

- When changing the number of processors needed for running a certain problem, make sure to change the number of blocks in the file PAR.F to match this number. Actually, any value for the number of blocks which is equal to or greater than the number of processors will suffice since the number of blocks is used only for memory allocation purposes. Therefore, if the number of blocks is larger than the number of processors, there will be some memory waste, but the code will run fine without the need for complete re-compilation if the number of processors is changed to any other value which is still less than the number of blocks given in the PAR.F file.

- The original RPLUS code was segmented into various *.f files for better readability and improved functionality. This also enhances efficiency since changes made to the files do not force a complete code recompilation. A complete set of these codes is included with this report in a hard disk as an attachment.

- The input2d.f file is a modified version of the original. The following changes must be remembered when running the parallel version of the code.
  1. A logical variable AUT_SPLIT was added to switch on the automatic domain decomposition option. If this variable is set to .TRUE., the user no longer needs to specify the values of IBEG, IEND, JBEG and JEND for any blocks. If a user defined decomposition is employed, this variable should be set to .FALSE. and IBEG, IEND, JBEG and JEND should be supplied for each block.
  2. In relation to #1, the arrays defining the boundary condition types on the edges of the blocks have been changed. The user now gives the boundary condition type only on the boundaries of the domain without reference to any particular block. The code will automatically map these values to the appropriate blocks and take care of the internal block interfaces. All interior block edges were assumed to have boundary condition type 6. The change is summarized below

Original	changed to	New
IFCBEG(J,NB)		IFCBEA(J)
IFDBEG(J,NB)		IFDBEA(J)
IFCEND(J,NB)		IFCENA(J)
IFDEND(J,NB)		IFDENA(J)

The original arrays shown above appear in the common block BNDIDX and their replacements can be found in BNDIDXA. Likewise, the common blocks BNDPPI and BNDPPJ are replaced by the common blocks BNDPPIA and BNDPPJA, respectively. Both the original and replacement common blocks remain in the code.

Run-Time Details

To compile and run the code using MPICH the following sequence must be followed after the above modifications have been accounted for.

1. make rplus2d : compiles the code
2. mpirun -np <number of processors> rplus2d : runs interactively while the system chooses the machines for running the code.

or

2b. mpirun -p4pg <file name> rplus2d : runs interactively but forcing the code to run on a list of machines given in the <filename>. The default <filename> is procgroup. A sample of procgroup is given below:

```
-------------------------------------------------------------------------
      lace01  0   /home1/fsnidal/APPL/PARA/rplus2d
      lace05  1   /home1/fsnidal/APPL/PARA/rplus2d
      lace08  1   /home1/fsnidal/APPL/PARA/rplus2d
      lace12  1   /home1/fsnidal/APPL/PARA/rplus2d
-------------------------------------------------------------------------
```

In this sample file, the executable rplus2d available in the directory /home1/fsnidal/APPL/PARA will run on the machines lace01, lace05, lace08 and lace12. The job has to be submitted from lace01 (the first entry in the first row which corresponds to a zero entry in the second column of the same row).

or

2c. bsub -n <# of proc.> mpichjob rplus2d : runs in batch mode

You have a choice in this approach of the number of processors. Note that you should give the full pathname to the executable when using LSF on the cluster. The file mpichjob is a simple script file to run batch jobs using MPICH under LSF. For more details, please check the location http://www.lerc.nasa.gov/WWW/ACCL/mpi.html. The file mpichjob is listed below

```
#! /bin/sh
#
# mpichjob
#
# This sample script is a wrapper for running MPICH jobs under lsbatch.
# Submit job by saying "bsub [otheroptions] -n k mpichjob command line",
# where k is the number of hosts to use, mpichjob is the name of this script,
# and command line is the command to run.  Note that you must use the
# full path name to your MPI program if it's not in your normal search path.
#
# e.g. bsub -n 3 mpichjob cpi
#
# Note the variable LSB_HOSTS is assigned by lsbatch system when this script
# is started by lsbatch.
#
# Written August 3, 1995

COMMANDLINE="$@"

####################
#Generate procgroup
####################

PROCGROUP=$HOME/.lsbatch/host$$.`hostname`
```

```
rm -f $PROCGROUP
nhosts=0
for word in $LSB_HOSTS
do
  if [ $nhosts -eq 0 ] ; then
    echo "local 0 $COMMANDLINE" >> $PROCGROUP
    FIRST=$word
  else
    echo $word "1 $COMMANDLINE" >> $PROCGROUP
  fi
  nhosts=`expr $nhosts + 1`
done

echo 'RUNNING ON' $FIRST
echo 'PROCGROUP START'
cat $PROCGROUP
echo 'PROCGROUP END'
echo

###################################
# run mpi job and save exit status
###################################
rsh $FIRST "$COMMANDLINE -p4pg $PROCGROUP"
exstat=$?

##################
# cleanup and exit
##################

rm -f $PROCGROUP
exit $exstat
```

Experimental Procedure

The performance indicator used here is the total execution time needed for running the code after establishing the connections among the different machines. This time includes two I/O periods: the first period is at the beginning of the code to read the grid file, block bounds and restart files if needed, while the second period is at the end of the code to write the output and restart files. According to the MPI 1.0 standard, there is no standard parallel I/O yet. However, it is hopefully going to be a part of MPI 2.0 which is expected to be released in the near future. Therefore, the I/O portions of the code were done in a serial manner where a master processor controls the

synchronization of the different messages. Even though the I/O part was done serially, it was found that this part takes a negligible amount of time (less than 0.1%) compared to the total running time. The execution time was measured using the MPI function MPI_WTIME( ) which gives the wall clock time between two different time stamps. For each number of processors, the test problem was run at least three times and then the execution times were averaged. Running the problem with varying number of processors, we can measure the speedup. Speedup for p processors is normally defined as

$$\text{Speedup} = \frac{\text{Execution time for 1 process}}{\text{Execution time for p prcocesses}}$$

$$\text{Parallelization Efficiency} = \frac{\text{Actual Speedup}}{\text{Ideal Speedup}} \quad \text{Where Ideal Speedup} = p$$

The parallelization efficiency was calculated as

All the experiments were conducted in a single user mode. The next section will discuss the results of some of these experiments.

Results

The results of this research are presented in the form of a supersonic flow at (M_∞ =4.0, P=0.01 atm) past a 10 degree half angle cone at zero incidence test case problem. Two grids were tested with a variety of processor configurations. Non-reacting flow simulations and simulations of the cone with an $H_2 - Air$ chemical reaction model are included. The data illustrate the flow field results obtained in both cases and present a variety of parallel statistics from runs on a Cray Y-MP and on the NASA Lewis Workstation Cluster known as Lewis Advanced Cluster Environment (LACE). The changes in parallelization efficiency due to the number of processors and the change in the convergence behavior of the scheme are illustrated.

Figure 3 demonstrates the near equality of the pressure contours obtained with the non-reacting computations of the cone obtained with single and multi-block computations. A **61X41** grid was

used. Small differences can be seen between the two results which were due to the block-to-block internal boundary point interpolation procedure existing in both versions of the code. This contention was verified by computing the multiple block case on both single and multiple processors, and the results were found to be identical. In addition to the differences in the equality of the results the convergence rate of the scheme decreases considerably as the number of domains increases, as illustrated in figure 2. However, it is remarkable to note that the convergence plots of the five runs eventually join together for $\|\rho u\|_2 \leq 3 \cdot 10^{-6}$, whereas, above this value the same parameter was consistently less steep as the number of processors increased. This result indicates a penalty for utilizing the parallel algorithm apart from any communication overhead as a result of simply splitting the problem up for parallelization. This issue was not explored further, but must be considered when one wishes to consider parallelizing a code. It is also important to recognize that the "level" to which one wishes to converge the solution is an issue in this regard, since convergence levels near machine zero are relatively unaffected by the parallelization.

Figures 4 and 5 show the different timing results for the cases and their percentage variations from the mean. Timings with and without the I/O portion are included. It is seen that results vary nearly $\pm 10\%$ from the mean value for the 61x41 grid test case. However, this number drops to less than $\pm 2\%$ from the mean value for the 121x81 grid, where block-to-block communication represents a smaller fraction of the total workload.

It should be noted that this test was made using only the Ethernet connection and not the other available communications ports. It was felt that this type of connection would be more representative of those testing workstation clusters for the first time, and would provide a more meaningful comparison.

Figures 6 and 7 illustrate the execution time versus number of processors for the 61x41 and 121x81 grids, respectively. These plots compare observed times versus those expected with ideal speed-up and versus results obtained on a single processor of a Cray Y-MP. The results clearly indicate that the actual execution time is slightly greater than the ideal, as expected. The reader will also see that in both cases approximately 4 LACE cluster workstations were needed to effectively match the performance of a single Y-MP processor. The results are slightly better for the larger grid size. We should state here though that the runs were performed using a single precision on both the workstations and the YMP. This is an unfair comparison since single precision on the YMP corresponds to double precision on the workstations (64 bit versus 32 bit.) However, an effort will be made to run the problems using double precision on the workstations to determine the effect on the timing results. It is expected that there will be minimal impact because the IBM machines do all arithmetic operations in double precision format. Hence, the only impact will be on communication overhead (which will double). However, this is a small fraction of the workload for the envisioned applications.

The improved performance experienced with the larger grid is further evident in figure 8, which plots for both grids the speed-up obtained with parallelization versus the ideal speed-up. These results are plotted in terms of a parallelization efficiency in figure 9. Both figures indicate that the smaller grid

is beginning to experience communication bottle necks when large numbers of processors are used.

It is important to note that although the LACE cluster is made up of 32 machines only 16 are the faster 590s. Figure 10 illustrates speed-up when the entire cluster is utilized with equal partition of work. Clearly this situation can be improved with a simple static load balancing approach that accounts for differences in processor speed. However, the current code does not yet do this automatically.

Another area requiring an improved load balancing technique are the reacting flow cases. Figure 11 contains the pressure contours for the same cone as above in an $H_2 - Air$ reacting mixture with a stoichiometric air to fuel ratio and $T_\infty = 1200\ K^0$. The reacting flow option was tested late in this research and parallelization statistics are not yet available.

In summary, the above results demonstrate the effectiveness of the parallel RPLUS2D code for non-reacting calculations. Improvements to the scheme are still required and are being pursued. Parallelization efficiencies greater than 84% were achieved with the larger grid size and indicate that the technique would be more useful for even larger grids. No attempt was made to study the effect of different block sizes, shapes or orientations. Convergence rate degradation was experienced but was not a factor if the code was converged near to machine zero. Specific convergence performance is therefore subject to desired convergence level.

Additional Modifications

Apart from the changes made to allow parallelization and improve implementation efficiency, a modification was made to the basic solver to install Roe's flux difference splitting as a solver option. The reason for this is that it was found that the Van Leer's flux vector splitting currently installed in the code is too dissipative especially when contact discontinuities are encountered. A shock-boundary layer interaction problem was run with both the Van Leer's flux splitting technique and the Roe's flux difference splitting technique. Much more accurate results were obtained with the Roe's solver especially within the separation zone inside the boundary layer.

Recommendations

Several issues have not been addressed in the current work and should be pursued for more useful application of the parallel RPLUS2D code.

- More efficient use of memory should be made. Currently, memory locations are included for the entire grid on each processor. This must be changed so that each processor reserves memory for only those data on which it will operate.
- A study should be made of the effect on the convergence rate and solution of grid block orientation. It is possible that partitioning in only 1 direction might enhance the convergence rate, i.e. if a dominant flow direction is present (as with boundary layers) it might be possible to decompose the solution domain such that block interfaces are aligned with and do not cross this direction.

- A dynamic load balancing approach needs to be incorporated to achieve efficient load balancing on heterogeneous clusters, multiuser environments, and flow fields with an uneven distribution of chemical reaction source terms work. An historical approach is possible for this need.
- Additional test cases should be computed to assess the ability of the parallel code to perform.
- Studying the performance of the different communication networks available on the cluster especially when considering larger problem sizes and having communication bottlenecks.
- Running the parallel version on different architecture such as the Cray T3D, IBM SP2 and assessing the performance on these architectures.

Project Personnel

The following personnel were funded by the research.

- Dr. Paul D. Orkwis (PI)
- Mr. Daniel B. Kim
- Mr. Nidal Ghizawi

Mr. Kim left the university before completing his degree. Mr. Ghizawi is a Ph.D. candidate under the co-direction of Dr. Orkwis and Dr. Abdallah. Unfortunately, considerable amounts of data and code were lost in the transition between the two students.

References

1. Tsai, Y.-L. P., "Recent Update of the RPLUS 2D/3D Codes," AIAA Paper 91-0094, 1991.

2. Hsiegh, K.C. and Tsai, Y.-L. P., "Comparative Study of Computational Efficiency of Two LU Schemes for Non-Equilibrium Reacting Flows," AIAA Paper 90-0396, 1990.

3. Keyes, D., "Domain Decomposition: A Bridge Between Nature and Parallel Computers," Adaptive, Multilevel, and Hierarchical Computational Strategies, ASME, AMD, Vol. 157, ASME, New York, NY, pp. 293-334.

4. Zapach, T.G., and Djilali, N., "Study of Accuracy and Parallel Efficiency of Domain Decomposition Applied to a Finite Volume Method," Advances in Computational Methods in Fluid Dynamics, ASME, FED, Vol. 196, 1994, ASME, New York, NY, pp 167-176.

5. Shimano, K., and Arakawa, C., "Numerical Simulation of Incompressible Flow on a Parallel Computer with the Domain Decomposition Technique," Transactions of the Japan Society of Mechanical Engineers, Part B, Vol. 59, No. 567, November 1993, pp. 3340-3346.

6. Drikakis, D., and Schreck, E., "Parallel Multi Level Calculations for Viscous Compressible Flows," CFD Algorithms and Applications for Parallel Processors, ASME, FMD, Vol. 156, 1993, ASME, New York, NY, pp. 9-23.

7. Schreck, E., and Peric, M., "Computation of Fluid Flow With a Parallel Multigrid Solver," International Journal for Numerical Methods in Fluids, Vol. 16, No. 4, February 1993, pp. 303-327.

8. Bhogeswara, R., and Killough, J.E., "Domain Decomposition and Multigrid Solvers for Flow Simulations in Porous Media on Distributed Memory Parallel Processors," Journal of Scientific Computing, Vol. 7, No. 2, June 1992, pp. 127-162.

9. Ewing, R. E., "Survey of Domain Decomposition Techniques and Their Implementation," Advances in Water Resources, Vol. 13, No. 3, September 1990, pp. 117-125.

10. Braaten, M. E., "Solution of Viscous Fluid Flows on a Distributed Memory Concurrent Computer," International Journal for Numerical Methods in Fluids," Vol. 10, No. 8, June 1990, pp. 889-905.

Figure Captions

Figure 1 Flow Chart of the New RPLUS Code.

Figure 2 Comparison of pressure contours for single and multi-block domain decompositions. Non-reacting mixture.

Figure 3 Norm-2 of $\Delta \rho u$ for the 61x41 grid obtained with various processors.

Figure 4 Variation in Time Measurements, 61x41 grid.

Figure 5 Variation in Time Measurements, 121x81 grid.

Figure 6 Execution Time for the 61x41 Grid.

Figure 7 Execution Time for the 121x81 Grid.